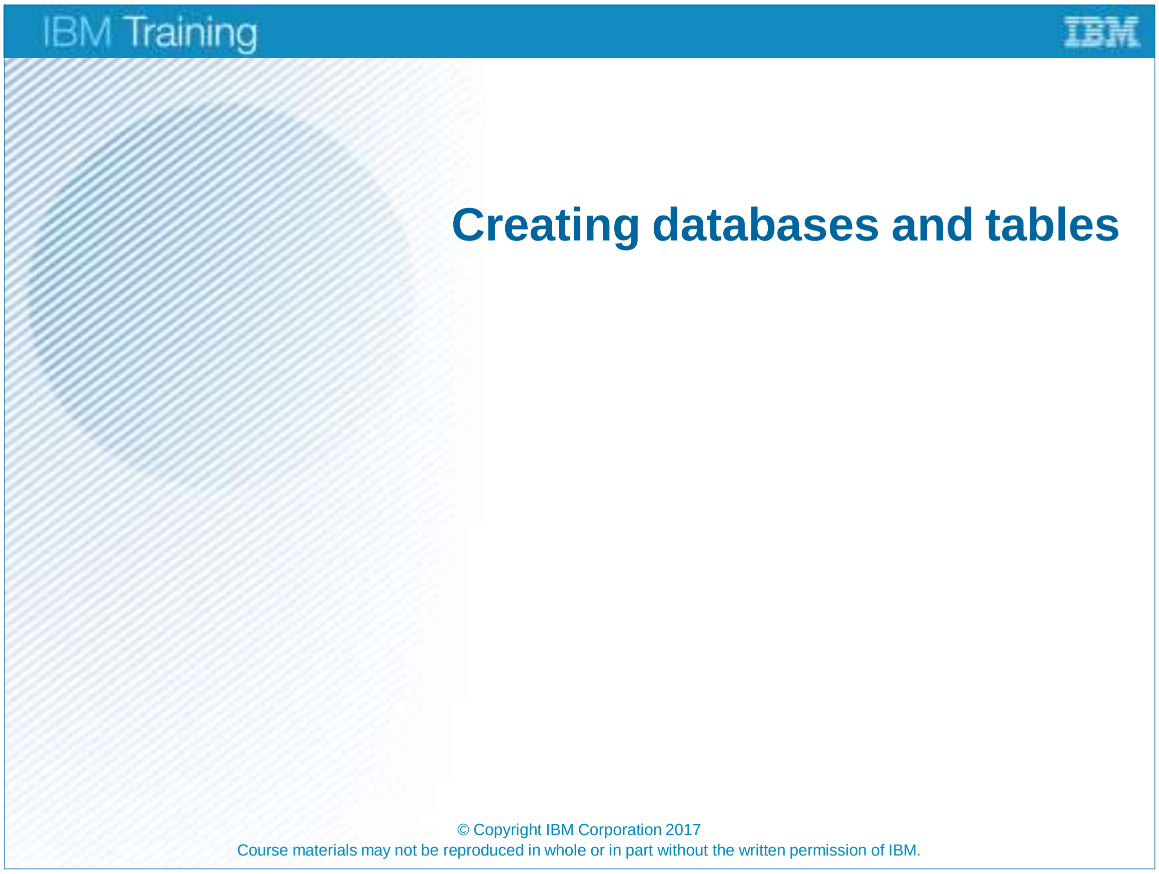

IBM Training

Creating databases and tables

© Copyright IBM Corporation 2017
Course materials may not be reproduced in whole or in part without the written permission of IBM.

Exercise 1

Create databases and tables

- create databases in specific locations
- create regular and temporary tables
- create tables for storing large objects
- use the dbschema utility for generating database and table schemas

© Copyright IBM Corporation 2017

Exercise 1:
Create databases and tables - Solutions

Purpose:
In this exercise, you will create databases and tables, including tables with large object data types. You will also use the dbschema utility for generating database and table schemas.

Task 1. Access the Informix system.

Task 2. Using dbaccess.

In this task, you will familiarize yourself with the dbaccess functions for entering, running, saving, and loading SQL statements.

1. At the UNIX prompt, enter the following command:

   dbaccess

   This will open the dbaccess editior, with the menu at the top.
2. From the dbaccess menu, choose **Query-language**. You can either highlight it and press enter, or just type in the letter q. The SELECT DATABASE prompt will be displayed.
3. At the **SELECT DATABASE** prompt, select **sysmaster@dev** by highlighting it and pressing **Enter**. The SQL menu will be displayed.
4. From the **SQL** menu, choose **New**. You can either highlight it and press Enter, or just type in the letter n. The cursor will move to the typing area.
5. In the typing area, type in the following SQL statement:

   SELECT * FROM SYSTABLE;

   Refer to the top of the screen for editing options (ESC done editing, CTRL-X delete a character, and so on.
6. After entering the SQL, press the **ESC** key on the keyboard. The SQL menu will be displayed.
7. To execute the SQL, choose **Run**. You can either highlight it and press enter, or just type in the letter r.
8. The SQL will be executed. Since it includes an error, an error message will be displayed at the bottom of the screen. (There is no table in the database named systable.) To correct the error, choose **Modify** from the menu. You can either highlight it and press Enter, or just type in the letter **m**.

9. The cursor will be displayed in the typing area, as close to the error as possible. Refer to the MODIFY menu at the top of the screen for editing options. Correct the error so the SQL reads as follows:

   SELECT * FROM SYSTABLES;
10. Press the **ESC** key on the keyboard. The SQL menu will be displayed.
11. Execute the SQL statement by choosing **Run**. If there are no further errors, the first page of the results will be displayed.
12. Page through the output by choosing **Next** from the DISPLAY menu. You can either highlight it and press enter, or just type in the letter n.
13. Continue paging through the output. When you have seen enough, choose **Exit** from the **DISPLAY** menu. You can either highlight it and press enter, or just type the letter e. The SQL menu will be displayed.
14. To save the SQL, choose **Save** from the **SQL** menu. You can either highlight it and press Enter, or just type the letter **s**. The SAVE>> prompt will be displayed.
15. At the **SAVE>>** prompt, type in the filename into which you want to save the SQL statement, in this case EXAMPLE, and then press **Enter**. The SQL menu will be displayed.
16. To exit out of the editing options, choose the **Exit** option from the **SQL** menu. You can either highlight it and press **Enter**, or just type the letter **e**. The DBACCESS menu will be displayed.
17. To exit out of **dbaccess**, choose the **Exit** option from the SQL menu. You can either highlight it and press **Enter**, or just type the letter **e**. The Unix prompt will be displayed.
18. Enter **dbaccess** again. Choose the **sysmaster** database. Choose the **Query-language** option. The SQL menu will be displayed.
19. To open an existing SQL file, select the **Choose** option from the **SQL** menu. You can either highlight it and press **Enter**, or just type the letter **c**. A list of your saved files will be displayed.
20. From the file list, choose the EXAMPLE file by highlighting it and pressing **Enter**. The contents of the file will be loaded into the typing area. Use the previously discussed commands to run the SQL and view the output.
21. Exit **dbaccess** and return to the **UNIX** prompt.

Task 3. Create a database.

In this task, you will create a database, identify its default dbspace location using oncheck, drop the database, and recreate it in a specified dbspace.

1. Create a database called **stores_demo** and have it use buffered logging.

   CREATE DATABASE stores_demo WITH BUFFERED LOG

2. Use **oncheck -pe** to locate the dbspace where your database has been created.

   oncheck -pe | more

   Is the database in the best location?

   No, the database is located in the rootdbs, where the system databases used to maintain the database server are stored.

   How do you specify the location of your database?

   Use the IN dbspace_name clause with your CREATE DATABASE statement.

3. **Drop** your database using the following statement (you must have another database selected first, to remove the 'stores_demo' database):

   DROP DATABASE stores_demo;

4. Recreate your database in the **dbspace1** dbspace. Remember to have it use buffered logging. Hint: Use the IN dbspace_name clause with your CREATE DATABASE statement.

   CREATE DATABASE stores_demo IN dbspace1 WITH BUFFERED LOG;

5. Use **oncheck -pe** to locate the dbspace where your database has been created.

   $ oncheck -pe dbspace1 | more

Task 4. Create tables.

In this task, you will create and load four tables in your database using specified dbspaces. You will use the oncheck utility to query the system catalogs to see how extents were allocated. You will drop all the tables, calculate the proper extent size usage, and recreate and reload the tables.

1. Make sure that you are in the **/home/informix/labs** directory. Create the SQL scripts in individual files so you can save and modify them later.

   ```
   $ cd /home/informix/labs
   ```

   From this point in the Exercise, coding is done in ‘dbaccess’

2. Create a **customer** table (type the code and Run) in your database with the following columns:

Column name	Description
customer_num	Number starting at 101 and increasing by 1 for every customer.
fname	Customer first name. Allow for a length of 15.
lname	Customer last name. Allow for a length of 15.
company	Company name. Allow for a length of 20.
address1	First address line for the customer. Allow for a length of 20.
address2	Second address line for the customer. Allow for a length of 20.
city	City where the customer lives. Allow for a length of 15.
state	State where the customer lives. Allow for a length of 2.
zipcode	Customer zipcode. Allow for a length of 5.

```
CREATE TABLE customer (
customer_num        SERIAL(101),
fname               CHAR(15),
lname               CHAR(15),
company             CHAR(20),
address1            CHAR(20),
address2            CHAR(20),
city                CHAR(15),
state               CHAR(2),
zipcode             CHAR(5)
)
;
```

3. Create a **stock** table (type the code and Run) in your database with the following columns:

Column name	**Description**
stock_num	Manufacturer stock number that identifies the specific item. It is a number, no greater than 1,000.
manu_code	Manufacturer code for the item. The code is 3 characters in length.
description	Item description. Allow for a length of 15.
unit_price	Item price per unit. It has a maximum of 6 digits, including 2 decimal places.
unit	Unit by which the item is ordered. Allow for a length of 4.
unit_descr	Description of the unit. Allow for a length of 15.

```
CREATE TABLE stock (
stock_num          SMALLINT,
manu_code          CHAR(3),
description        CHAR(15),
unit_price         MONEY(6,2),
unit               CHAR(4),
unit_descr         CHAR(15)
)
;
```

4. Create an **orders** table (type the code and Run) in your database with the following columns:

Column name	**Description**
order_num	Number starting at 1001 and increasing by 1 for every order.
order_date	Date the order is placed.
customer_num	Customer number this order belongs to. This refers to the customer number in the customer table.
po_num	Customer purchase order number. It can contain letters and numbers. Allow for a length of 10.
ship_date	Date the order is shipped.
ship_weight	Shipping weight of the order. It contains a maximum of 8 digits, including 2 decimal places.
ship_charge	Shipping charge amount. It contains a maximum of 6 digits, including 2 decimal places.
paid_date	Date the order is paid.

```
CREATE TABLE orders (
order_num          SERIAL(1001),
order_date         DATE,
customer_num       INTEGER,
po_num             CHAR(10),
ship_date          DATE,
ship_weight        DECIMAL(8,2),
ship_charge        MONEY(6,2),
paid_date          DATE
)
;
```

5. Create an **items** table (type the code and Run) in your database with the following columns:

Column name	**Description**
item_num	Numeric identifying the individual line number of this item. It has a maximum value of 1,000.
order_num	Order number this item belongs to. This refers to the order number in the orders table.
stock_num	Stock number for the item. This refers to the stock number in the stock table.
manu_code	Manufacturer code for the item ordered. This refers to the manufacturer code in the stock table.
quantity	Quantity ordered. This has a maximum value of 2,000.
total_price	Quantity ordered times unit price (from the stock table). Allow for a maximum of 8 digits including 2 decimal places.

```
CREATE TABLE items (
item_num          SMALLINT,
order_num         INTEGER,
stock_num         SMALLINT,
manu_code         CHAR(3),
quantity          SMALLINT,
total_price       MONEY(8,2)
)
;
```

6. Run the load script **load.sql**. This script loads the tables you created. Use the following command to execute the load script:

   ```
   $ dbaccess stores_demo load.sql
   ```

   Use **oncheck -pe** to determine the location of the table extents. Save the output to a file to compare later.

   Hint: If you do not specify the dbspace when creating a table, where is the table stored? **It is stored in the same dbspace as the database** (in this case, dbspace1).

   Hint: When using oncheck to create an output file of your tables for a specific dbspace, use the following command:

   $ oncheck -pe *dbspace_name* > *filename*

   For example:

   $ oncheck –pe dbspace1 > D1T3S6 (represents Demo 1, Task 3, Step 6)

Have the tables been efficiently located and loaded in the best possible manner?

No, the tables are all stored in dbspace1, and may have multiple fragments interleaved with fragments of other tables.

Since your tables are already created, you can query **systables** table to get information that is needed to calculate extent sizes. What information would that be? Note: nrows may show up as 0 since UPDATE STATISTICS has not been run yet.

Rowsize, number of columns, and number of rows have information that help in calculating the extent sizes.

```
SELECT rowsize, ncols, nrows
from systables
where tabname = 'table name';
```

7. Drop each of your tables using the following command:

   DROP TABLE table_name;

8. Use the following information to recreate your **customer** table:

   - Calculate the first extent size to initially store 1200 rows.
   - Calculate a next extent size assuming that your table will grow approximately 10 percent per year for one year.
   - Use row-level locking.
   - Locate the table in **dbspace2**.

```
CREATE TABLE customer (
customer_num        SERIAL(101),
fname               CHAR(15),
lname               CHAR(15),
company             CHAR(20),
address1            CHAR(20),
address2            CHAR(20),
city                    CHAR(15),
state               CHAR(2),
zipcode             CHAR(5)
)
IN dbspace2
EXTENT SIZE 150 NEXT SIZE 16
LOCK MODE ROW;
```

The name, company, and address columns can be either CHAR or VARCHAR. If the column is likely to be indexed, it is best to use a CHAR data type.

Assuming the structure given above and a 2-kilobyte page size, the initial extent calculation is:

-Initial rows = 1200

-Rowsize = 4 + 15 + 15 + 20 + 20 + 20 + 15 + 2 + 5 + 4 (slot) = 120

-Pageuse = 2048 - 28 = 2020

Since rowsize is less than the page size:

-Rows per page = (pageuse/rowsize) = 2020 / 120 = 16.83 ==> 16

-Number of data pages required = 1200 / 16 = 75

-Number of kilobytes required = (75 * 2048) / 1024 = 150

The EXTENT SIZE is 150.

-Assuming that the table will grow by 10 percent per year, the DBA can calculate the NEXT SIZE by applying the calculations above for growth rows:

-Number of rows at 10 percent growth = 1200 / 10 = 120

-Number of data pages required = 120 / 16 = 7.5 ==> 8

-Number of kilobytes required = (8 * 2048) / 1024 = 16

The NEXT SIZE is 16.

9. Use the following information to recreate your **orders** table:

   - Calculate the first extent size to initially store 500 rows.
   - Calculate a next extent size assuming that your table will grow approximately 10 percent per year for one year.
   - Use row-level locking.
   - Locate the table in **dbspace3**.

```
CREATE TABLE orders (
order_num           SERIAL(1001),
order_date          DATE,
customer_num        INTEGER,
po_num              CHAR(10),
ship_date           DATE,
ship_weight         DECIMAL(8,2),
ship_charge         MONEY(6,2),
paid_date           DATE
)
IN dbspace3
EXTENT SIZE 22 NEXT SIZE 8
LOCK MODE ROW;
```

Assuming the structure given above and a 2-kilobyte page size, the initial extent calculation is:

-Initial rows = 500

-Rowsize = 4 + 4 + 4 + 10 + 4 + 5 + 4 + 4 + 4 (slot) = 43

-Pageuse = 2048 - 28 = 2020

Since rowsize is less than the page size:

-Rows per page = (pageuse/rowsize) = 2020 / 43 = 46.97 ==> 46

-Number of data pages required = 500 / 46 = 10.869 ==> 11

-Number of kilobytes required = (11 * 2048) / 1024 = 22

The EXTENT SIZE is 22.

Assuming that the table will grow by 10 percent per year, the DBA can calculate the NEXT SIZE by applying the calculations above for growth rows:

-Number of rows at 10 percent growth = 500 / 10 = 50

-Number of data pages required = 50 / 46 = 1.0869 ==> 2

However, the minimum extent size is 4 pages, so the NEXT SIZE must be:

-Number of kilobytes required = (4 * 2048) / 1024 = 8

The NEXT SIZE is 8.

10. Use the following information to recreate your **items** table:
    - Calculate the first extent size to initially store 500 rows.
    - Calculate a next extent size assuming that your table will grow approximately 10 percent per year for one year.
    - Use row-level locking.
    - Locate the table in **dbspace4**.

```
CREATE TABLE items (
item_num       SMALLINT,
order_num      INTEGER,
stock_num      SMALLINT,
manu_code      CHAR(3),
quantity       SMALLINT,
total_price    MONEY(8,2)
)
IN dbspace4
EXTENT SIZE 12 NEXT SIZE 8
LOCK MODE ROW;
```

Assuming the structure given above and a 2-kilobyte page size, the initial extent calculation is:

-Initial rows = 500

-Rowsize = 2 + 4 + 2 + 3 + 2 + 5 + 4 (slot) = 22

-Pageuse = 2048 - 28 = 2020

Since rowsize is less than the page size:

-Rows per page = (pageuse/rowsize) = 2020 / 22 = 91.81 ==> 91

-Number of data pages required = 500 / 91 = 5.494 ==> 6

-Number of kilobytes required = (6 * 2048) / 1024 = 12

The EXTENT SIZE is 12.

Assuming that the table will grow by 10 percent per year, the DBA can calculate the NEXT SIZE by applying the calculations above for growth rows:

-Number of rows at 10 percent growth = 500 / 10 = 50

-Number of data pages required = 50 / 91 = 0.549 ==> 1

However, the minimum extent size is 4 pages, so the NEXT SIZE must be:

-Number of kilobytes required = (4 * 2048) / 1024 = 8

The NEXT SIZE is 8.

11. Use the following information to recreate your **stock** table:

- Calculate the first extent size to initially store 74 rows.
- Calculate a next extent size assuming that your table will grow approximately 10 percent per year for one year.
- Use row-level locking.
- Locate the table in **dbspace4**.

```
CREATE TABLE stock (
stock_num      SMALLINT,
manu_code      CHAR(3),
description    CHAR(15),
unit_price     MONEY(6,2),
unit           CHAR(4),
unit_descr     CHAR(15)
)
IN dbspace4
LOCK MODE ROW;
```

Assuming the structure given above and a 2-kilobyte page size, the initial extent calculation is:

-Initial rows = 74

-Rowsize = 2 + 3 + 15 + 4 + 4 + 15 + 4 (slot) = 47

-Pageuse = 2048 - 28 = 2020

Since rowsize is less than the page size:

-Rows per page = (pageuse/rowsize) = 2020 / 47 = 42.98 ==> 42

-Number of data pages required = 74 / 42 = 1.761 ==> 2

-Number of kilobytes required = (2 * 2048) / 1024 = 4

However, the minimum extent size is 4 pages, so the EXTENT SIZE must be:

-Number of kilobytes required = (4 * 2048) / 1024 = 8

The EXTENT SIZE is 8.

Assuming that the table will grow by 10 percent per year, the DBA can calculate the NEXT SIZE by applying the calculations above for growth rows:

-Number of rows at 10 percent growth = 74 / 10 = 7.4

-Number of data pages required = 7.4 / 42 = 0.176 ==> 1

However, the minimum extent size is 4 pages, so the NEXT SIZE must be:

-Number of kilobytes required = (4 * 2048) / 1024 = 8

The NEXT SIZE is 8.

Notice that the EXTENT SIZE and NEXT SIZE have not been included in the CREATE TABLE syntax. Why?

The default size for table extents is 8 pages. Therefore, on a system with a 2-kilobyte page size, the default sizes for EXTENT SIZE and NEXT SIZE would be 16 kilobytes each. This is plenty of room to store the rows for the stock table and, therefore, the explicit inclusion of EXTENT SIZE and NEXT SIZE is not required.

12. Use the load script to load your tables again.

    $ dbaccess stores_demo load.sql

13. Run the following commands on each table:

    Use **oncheck -pe** to check your table to see how the extents were allocated. Compare how extents were allocated when defaults were used compared to their current layout.

    $ oncheck -pe *dbspace_name* > *filename*

    For example:

    $ oncheck –pe dbspace1 > D1T3S6 (represents Demo 1, Task 3, Step 13)

    - Query the **systables** system catalog for the **customer** table for the following:
      - First extent size
      - Next extent size
      - Type of table locking
      - Row size

    Is the row size correct? Why or why not?

    The rowsize is correct for the data columns, but it does not include the extra four bytes for the slot table entry.

    ```
    SELECT * FROM systables
    WHERE tabname = "customer";
    ```

Task 5. Create a table for a simple large object.

You will create and load a table for a simple large object in a specified dbspace.

1. Create a **catalog** table in your database in the **dbspace4** dbspace with the following columns:

Column Name	Description
catalog_num	Number starting at 10001 and increasing by 1 for every catalog entry.
stock_num	Stock number for the item. This refers to the stock number in the **stock** table.
cat_descr	Description of item. This is a text description and can be large.
cat_picture	Picture of item. This is an image.
cat_advert	Tag line below the picture. The tag line can be as many as 255 characters long, but is never fewer than 65.

   Put the simple large objects in the table. Be sure and calculate the extent sizes as in the previous task, allowing for 74 rows.

```
CREATE TABLE catalog (
catalog_num   SERIAL(10001),
stock_num     SMALLINT,
cat_descr     TEXT,
cat_picture   BYTE,
cat_advert    VARCHAR(255,65)
)
IN dbspace4
EXTENT SIZE 30 NEXT SIZE 8
LOCK MODE ROW;
```

Assuming the structure given above and a 2-kilobyte page size, the initial extent calculation is:

Initial rows = 74

Rowsize = 4 + 2 + 56 + 56 + 256 + 4 (slot) = 378

Use the maximum length value for the cat_advert column. In this case it is 255 bytes, plus 1 for the length byte for the VARCHAR.

Pageuse = 2048 - 28 = 2020

Since rowsize is less than the page size:

Rows per page = (pageuse/rowsize) = 2020 / 378= 5.34 ==> 5

Number of data pages required = 74 / 5= 14.8 ==> 15

Number of kilobytes required = (15 * 2048) / 1024 = 30

The EXTENT SIZE is 30;

Assuming that the table will grow by 10 percent per year, the DBA can calculate the NEXT SIZE by applying the calculations above for growth rows:

Number of rows at 10 percent growth = 74 / 10 = 7.4

Number of data pages required = 7.4 / 5 = 1.48 ==> 2

However, the minimum extent size is 4 pages, so the NEXT SIZE must be:

Number of kilobytes required = (4 * 2048) / 1024 = 8

The NEXT SIZE is 8.

This only calculates the extent sizes for the data rows, and does not include the storage space required for the cat_descr and cat_picture objects themselves. These objects will be stored in the same tablespace as the data rows, so their size estimates must also be included.

2. Load the **catalog** table using the load script **loadcat.sql**.

   ```
   $ dbaccess stores_demo loadcat.sql
   ```

Task 6. Create a temporary table.

In this task, you will be using two windows: one window will be used to create a temporary table (and write scripts) and the other to monitor the temporary table. First, however, you must make sure that the database engine is configured to recognize the temporary dbspaces.

1. Bring the engine offline by issuing the following command:

   $ onmode -ky

2. Edit the **onconfig** file (using the vi editor) and make sure that the DBSPACETEMP parameter has your two temporary dbspaces listed as shown below. Important: Do not put any spaces or other white space in the list of dbspace names.

   DBSPACETEMP   tempdbs1:tempdbs2

   $ vi $INFORMIXDIR/etc/$ONCONFIG

3. Bring the engine online by issuing the following command:

   $ oninit

4. Open (and log into = **docker/tcuser**) a 2nd PUTTY session window. Run the command: **docker exec -it iif_developer_edition bash**. In the first window, open a dbaccess session and create a temporary table called **cust_temp.** Make it a no logging table. The table contains **customer_num, fname,** and **lname** from the **customer** table.

   Important: Do not exit this dbaccess session. You will continue with it in a later exercise.

   SELECT customer_num, fname, lname

   FROM customer

   INTO TEMP cust_temp WITH NO LOG;

5. In the second window, run **oncheck -pe** to see the temporary table usage.

   $ oncheck -pe tempdbs1

   The oncheck -pe tempdbs1 output displays the dbspace mapping with the cust_temp temporary table using 16 pages in the tempdbs1 temporary dbspace.

Task 7. Use dbschema.

In this task, you will create files containing the statements to recreate the tables from your database. Be sure to save your SQL scripts so you can use them again in future tasks.

Use the **dbschema** utility to create the following files:

1. From the second window, create a file called **customer_schema.sql**, which will contain the SQL statements necessary to recreate the **customer** table. Be sure to use the -ss option to capture lock level and dbspace information.

   ```
   $ dbschema -d stores_demo -t customer -ss customer_schema.sql
   ```
2. Create a file called **orders_schema.sql,** which will contain the SQL statements necessary to recreate the **orders** table.

   ```
   $ dbschema -d stores_demo -t orders -ss orders_schema.sql
   ```
3. Create a file called **items_schema.sql,** which will contain the SQL statements necessary to recreate the **items** table.

   ```
   $ dbschema -d stores_demo -t items -ss items_schema.sql
   ```
4. Create a file called **stock_schema.sql,** which will contain the SQL statements necessary to recreate the **stock** table.

   ```
   $ dbschema -d stores_demo -t stock -ss stock_schema.sql
   ```
5. Create a file called **catalog_schema.sql,** which will contain the SQL statements necessary to recreate the **catalog** table.

   ```
   $ dbschema -d stores_demo -t catalog -ss catalog_schema.sql
   ```

Results:
In this exercise, you created databases and tables, including accounting for large object data types. You also used the dbschema utility for generating database and table schemas.